逐次型推論マシンＣＨＩ小型化版のハードウェア

5B-8

幅田伸一　、中崎良成　、梅村　護
日本電気（株）Ｃ＆Ｃシステム研究所

はじめに　通産省第５世代計算機プロジェクトの一環として、高性能Ｐｒｏｌｏｇマシンｃｈｉ［1］の小型化版を開発中である。小型化版ＣＨＩは、使用素子をＣＭＬからＴＴＬとＣＭＯＳに変更し、１メガビットＤＲＡＭを採用することで、デスクサイド・タイプの計算機システムにする予定である。しかし、使用素子を高速のＣＭＬからＴＴＬとＣＭＯＳに変更したため、マシン・サイクル・タイムが増加した。マシン・サイクル・タイムの増加は、小型化版ＣＨＩの性能を低下する。本稿では、ＣＭＬ版ＣＨＩと同性能にすることを目標に、小型化版ＣＨＩで新たに導入したハードウェア機能について報告する。

小型化の概要　小型化版ＣＨＩは、ＣＭＯＳゲートアレイによるＬＳＩ化と、ＣＭＬ素子より集積度の高いＴＴＬ素子の使用により、プロセッサ部を基板４枚にしている。また、主記憶部は、１メガビットＤＲＡＭの採用と、高密度実装技術の導入により、基板１８枚にしている。ＣＭＬ版ＣＨＩと小型化版ＣＨＩのハードウェア構成の比較を第１表に示す。

小型化により発生した問題点　高速のＣＭＬ素子からＴＴＬとＣＭＯＳ素子への変更は、ゲート当たりの遅延時間増加を意味する。プロセッサ部の主記憶アクセス速度を改善するキャシュメモリのアクセス時間で比較すると、ＣＭＬ素子使用のものに比べ２倍の時間が必要となる。ゲート遅延時間とキャシュメモリのアクセス時間を中心にＣＭＬ素子とＴＴＬ，ＣＭＯＳ素子を比較した結果を第２表に示す。

キャシュメモリのアクセス時間から分るように、小型化版ＣＨＩでは、ＣＭＬ版ＣＨＩのマシン・サイクル・タイムを実現できない。

マシン・サイクル・タイム増加の対策　ＣＨＩが実行対象としているＰｒｏｌｏｇは、実行環境の生成、消去を頻繁に行う。これらの処理は、メモリ・アクセスを伴う処理であり、メモリ・アクセス時間が、処理時間に与える影響は大きい。小型化版ＣＨＩでは、マシン・サイクル・タイムをキャシュメモリのアクセス時間に近づけ、メモリ・アクセス時間の増加による性能低下を最小にするように努めている。この結果、小型化版ＣＨＩのマシン・サイクル・タイムは、キャシュメモリのサイクル・タイムである１７０ｎＳｅｃとしている。さらに、１７０ｎＳｅｃのマシン・サイクル・タイム中で実現できるハードウェア機能を追加し、１つの機械語命令を実行するのに必要なマシン・サイクル数を減らしている。以下、性能改善の為に導入したハードウェア機能とその効果について説明する。

新ハードウェア機能　マシン・サイクル・タイムをキャシュメモリのアクセス時間から決定したので、小型化版ＣＨＩでは、各機械語命令の実行に必要なマイクロステップ数を、メモリ・アクセス回数に近づけるように努めている。この為に、小型化版ＣＨＩでは、

・機械語命令先取り用アドレス演算の一部をハードウェア化
・４個のメモリ・アドレス・レジスタの設置
・タグを使用した条件分岐機能の強化

のハードウェア上の改造を行っている。

第１表　ＣＭＬ版ＣＨＩと小型化版ＣＨＩのハードウェア構成の比較

比較項目	ＣＭＬ版ＣＨＩ	小型化版ＣＨＩ
使用素子	ＣＭＬ 256 Kbit DRAM	ＴＴＬ ＣＭＯＳ 1 Mbit DRAM
基板枚数 プロッセサ部 主記憶部	 ８１枚 約１１０枚	 ４枚 １８枚
マシン・サイクル	１００ｎＳ	１７０ｎＳ
データ幅	３６ビット	４０ビット
主記憶容量	６４ＭＷ （２５６ＭＢ）	１２８ＭＷ （６４０ＭＢ）
キャシュメモリ容量	８ＫＷ （３２ＫＢ）	３２ＫＷ （１６０ＫＢ）
マイクロ命令幅	８０ビット	７８ビット
制御メモリ容量 （ＷＣＳ容量）	１１ＫＷ （１１０ＫＢ）	１６ＫＷ （約１６０ＫＢ）

第２表　ＣＭＬ素子とＴＴＬ，ＣＭＯＳ素子の比較

比　較　項　目	ＣＭＬ素子	ＴＴＬ，ＣＭＯＳ素子
ゲート遅延時間	７００ｐＳ	ＣＭＯＳ　２ｎＳ ＴＴＬ　３．５ｎＳ
キャシュメモリ アクセス時間	７５ｎＳ	１５０ｎＳ

Hardware Design of a CHI　Compact Version
Shin-ichi HABATA, Ryousei NAKAZAKI, Mamoru UMEMURA
NEC Corporation

機械語命令先取り用アドレス演算機能　ＣＨＩでは、Ｐｒｏｌｏｇの述語呼出し、引数、複合体の要素毎に、専用の機械語命令を用意している。この為、各機械語命令の処理は、２、３回のメモリ・アクセスを伴った簡単な処理が多い。従って、機械語命令の読み出しとＯＰコードの解読によるマイクロ・ステップ数の増加は、処理速度を著しく低下させる。この為、ＣＭＬ版ＣＨＩでも、機械語命令先取り用バッファを備えていた。［２］小型化版ＣＨＩでは、この機械語命令先取り用バッファの最適化と先取り機械語命令のアドレス演算機能のハードウェア化を行っている。アドレス演算ハードウェアにとって、分岐命令の処理が問題となる。小型化版ＣＨＩでは、無条件相対番地分岐命令の分岐先アドレス演算機能をハードウェア化している。このハードウェア化により、述語読み出し（ｃａｌｌ）、クローズのインデクシング（ｔｒｙ，ｒｅｔｒｙ，ｔｒｕｓｔ）系の命令のステップ数が減っている。その他の分岐命令に対するアドレス演算は、マイクロ制御のもとで行う。

メモリ・アドレス・レジスタの複数化　メモリ・アドレス・レジスタを複数化する事は、マシン・サイクル・タイムの増加につながる。しかし、アドレス・セット操作を省略できる場合が多く、マイクロステップ数減少による性能改善が期待できる為、小型化版ＣＨＩでは、メモリ・アドレス・レジスタを４個とする。この結果、構造体要素の処理用機械語命令（ｕｎｉｆｙ命令）が使用するポインタが、メモリ・アドレス・レジスタに常駐し、マイクロステップ数減少の効果を上げている。更に、機械語命令先取り用に、専用のアドレス・レジスタを用意する。このように、メモリ・アドレス・レジスタを４個にし、機械語命令先取り用アドレス・レジスタを設置したことは、メモリ・アドレス・レジスタにアドレスをセットするマイクロステップを取り除く効果がある。結果として、メモリ・アクセスを行うマイクロステップの割合が増加し、機械語命令の実行に必要なマイクロステップ数をメモリ・アクセス回数に近づけている。

タグ分岐機能の強化　タグ・アーキテクチャを採用しているＣＨＩでは、タグを使用した条件分岐機能が、性能に大きく貢献している。このタグを使用した条件分岐機能を、タグ分岐機能と呼んでいる。小型化版ＣＨＩでは、このタグ分岐機能を、単なるタグのチェックではなく、同時に、演算器を使用したバリュー部のチェックも含む機能に拡張する。拡張したタグ分岐機能は、述語呼出側と呼出された側の引数のタグの組合わせとバリュー部の比較結果による４方向の分岐機能である。この分岐機能の使用例を、第３表に示す。この拡張により、ユニフィケーション処理の成功／失敗の判定、変数同士のバインディング操作時のポインタの方向の決定に必要なマイクロステップ数が減少している。

小型化版ＣＨＩの性能　上記新ハードウェア機能の導入による性能改善の効果を機械語命令単位でもとめた結果を第４表に示す。マイクロステップ数は、ＣＭＬ版ＣＨＩより減少していることが分る。マシン・サイクル・タイムの増加を考慮しても、ＣＭＬ版ＣＨＩよりも処理時間が減少している機械語命令が存在する。この事は、応用プログラムにより、小型化版ＣＨＩでも、ＣＭＬ版と同程度の性能が期待できることを示している。

まとめ　使用素子の変更によるマシン・サイクル・タイムの増加は、小型化版ＣＨＩにとって、性能低下の要因であった。特に、ＣＨＩにおいて、メモリ・アクセス時間が性能に与える影響が大きいため、小型化版ＣＨＩでは、マシン・サイクル・タイムをキャシュメモリのアクセス時間に近づけ、マシン・サイクル・タイム増加によるメモリ・アクセス時間の悪化を小さくするように努めた。さらに、ハードウェア機能を強化することで、機械語命令の実行に必要なマイクロステップ数を、メモリ・アクセス回数に近づけ、マシン・サイクル・タイム増加の影響を軽減することに努めた。この結果、一部の機械語命令では、ＣＭＬ版ＣＨＩより高速になった。しかし、処理時間が増えた機械語命令も多く、全体の性能を評価する必要がある。

参考文献

［１］Nakazaki et al.;"Design of a High-Speed Prolog Machine" Computer Architecture,1985

［２］幅田　他；高性能ＰＲＯＬＯＧマシン（ＣＨＩ）の機械語命令先取り方式についての考察
情報処理学会第３２回全国大会

第３表　小型化版ＣＨＩのタグ分岐機能

タグの組合わせ	バリュー部の関係	分岐先の処理
atom = atom	等しい	次の機械語命令呼出し
	異なる	ｆａｉｌ処理
nil = nil	ーーー	次の機械語命令呼出し
lunb = lunb	Ｘａ ＞ Ｘｂ	変数Ｘａから変数Ｘｂへポインタを張る
	Ｘａ ＜ Ｘｂ	変数Ｘｂから変数Ｘａへポインタを張る

第４表　ＣＭＬ版ＣＨＩと小型化版ＣＨＩの機械語命令におけるマイクロステップ数の比較

機械語命令名	ＣＭＬ版ＣＨＩ	ＣＨＩ小型化版	メモリアクセス回数
get permanent variable	2	2	2
get temporary variable	2	1	1
get permanent value	5	3	2
get temporary value	3	1	1
get list	2	1	1
get structure	6	3	2
put permanent variable	3	2	2
put temporary variable	3	2	2
put permanent value	4	3	2
put temporary value	3	1	1
put list	3	1	1
put structure	5	2	2
unify permanent variable	4	3	3
unify temporary variable	4	2	2
unify permanent value	5	3	3
unify temporary value	5	2	2
unify list	2	1	1
unify structure	6	3	2